Class Responsibility Assignment (CRA) for Use Case Specification to Sequence Diagrams (UC2SD)

Jali, N., Greer, D., & Hanna, P. (2014). *Class Responsibility Assignment (CRA) for Use Case Specification to Sequence Diagrams (UC2SD)*. 13-18. Paper presented at IEEE 2014 8th Malaysian Software Engineering Conference, Langkawi, Malaysia. https://doi.org/10.1109/MySec.2014.6985982

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
© 2014 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Download date:06. May. 2023

Class Responsibility Assignment (CRA) for Use Case Specification to Sequence Diagrams (UC2SD)

Nurfauza Jali[1, 2], Des Greer[1] and Philip Hanna[1]
{njali01| des.greer| p.hanna}@qub.ac.uk

[1] School of Electronics, Electrical Engineering and Computer Science
Queens University Belfast
Belfast, Northern Ireland
United Kingdom

[2] Faculty of Computer Science and Information Technology
Universiti Malaysia Sarawak
Kota Samarahan, Sarawak
Malaysia

***Abstract*—Identifying responsibility for classes in object-oriented software design phase is a crucial task. This paper proposes an approach for producing high quality and robust behavioural diagrams (e.g. Sequence Diagrams) through Class Responsibility Assignment (CRA). GRASP or General Responsibility Assignment Software Pattern (or Principle) was used to direct the CRA process when deriving behavioural diagrams. A set of tools to support CRA was developed to provide designers and developers with a cognitive toolkit that can be used when analysing and designing object-oriented software. The tool developed is called Use Case Specification to Sequence Diagrams (UC2SD). UC2SD uses a new approach for developing Unified Modelling Language (UML) software designs from Natural Language, making use of a meta-domain oriented ontology, well established software design principles and established Natural Language Processing (NLP) tools. UC2SD generates a well-formed UML sequence diagrams as output.**

Keywords—Class Responsibility Assignment; Software model; UML; Software Design Pattern; Responsibility Driven Design

I. INTRODUCTION

Finding class responsibilities in object-oriented analysis and design (OOAD) is not an easy task, although there is evidence that responsibility driven approach is effective [1][2][3][4]. Not only is this crucial during early analysis and design phases, but also during maintenance when new responsibilities have to be assigned to classes, or existing responsibilities have to be changed. All the current approaches are depends on human interpretation and decision making. This paper addresses the need to assist this decision making and describes an approach that can be used to support and improve the translation of natural language based software requirements to behavioural models that can in turn be translated to program code.

A set of tools to support Class Responsibility Assignment (CRA) was developed to provide designers and developers with a cognitive toolkit that can be used when analysing and designing object-oriented software. This attempt is applied to a tool called Use Case Specification to Sequence Diagrams (UC2SD)[5]. UC2SD is a tool that uses a new approach for developing Unified

Modelling Language (UML) software designs from Natural Language, making use of a meta-domain oriented ontology, well established software design principles and Natural Language Processing (NLP) tools [6][7]. This tool then generates a well-formed UML sequence diagrams as an output.

The rest of the paper is organized as follows. In the next section, we provide background overview of related research. Section III presents the description technique of GRASP including responsibility description. We respectively present the developed tool and a demonstration of its efficacy wit in Section IV and provide some discussion and conclusions in section V.

II. BACKGROUND AND RELATED WORK

The most vital and important step in creating the object-oriented software design is assigning responsibilities to classes. The design problem relates with the difficulty in identifying the participating classes and instances, collaboration, responsibilities and their role. Thus, design patterns are useful because they could save time and effort in solving a problem that already been solved.

Furthermore, when the patterns' adoption increases among the designers, this in turns reduces their cognitive load when they encounter the similar design elements. In other word, pattern provides a blueprint that is very useful for the software designers. In OOAD, Class responsibility assignment (CRA) is known as a significant learning aid in helping to identify classes, responsibilities and roles. Bowman [8] and Larman [9] defines CRA as how the classes responsibilities can be identified through the form of class operations/methods and as well as the manipulated attributes belonging to and how object should interact by using those operations. Larman has stated that "the critical design tool for software development is a mind well educated in design principles. It is not the UML or any other technology." The designers must have knowledge and skills in understanding the design patterns by mapping the problem and provides guidelines for imaging and designing the solution.

There has not been much work related to responsibilities of classes to improve the quality of software design. We are aware that there exist few well-reasoned and described design patterns or principles that assist us in understanding the essential object and class design [4]. There are Responsibility Driven Design (RDD)[10], Gang of Four (GoF) [11], and

GRASP[9]. In this paper, we only focus on the GRASP pattern because it is the basis of RDD and more generic than GoF. Larman has introduced GRASP or also known as General Responsibility Assignment Software Patterns or Principles as a cognitive toolset, which consist of guidelines for assigning responsibility to classes and objects in object-oriented design. GRASP guidelines help software designers to balance the trade-offs and give advantages for writing class methods with behaviours that affect multiple classes. In solving the CRA problem, there were current works done by Bowman and Glavas by using the Genetic algorithm[6][10] and Metaheuristic approach [11][12]. Some of their approaches have been adapted and applied in this project in much more simplified manner.

III. PROPOSED APROACHES

We have proposed architecture for a toolset, Use Case specification to Sequence Diagrams (UC2SD) that allows us to produce UML sequence diagrams from the text requirements provided by the stakeholder(s). This architecture will focus on the modelling aspects of the process, largely where the result is to generate diagrams and software code to represent the solution. Requirements analysis will then include an automated Natural Language Processing (NLP) [10][11] process. In turn, the requirements analysis to design phase will involve the extraction of object model elements such as classes, attributes, methods and relationships derived from the NLP [12][13]. The inclusion of knowledge in related to domain ontologies that will help to refine the object and properties candidates. In the software design and the implementation phases, these components will assist in building software models such as UML diagrams and software code. The data verification for each module will be evaluated by human experts. Data correction is a part of the verification process. Thus, we are aiming at design support rather than complete automation.

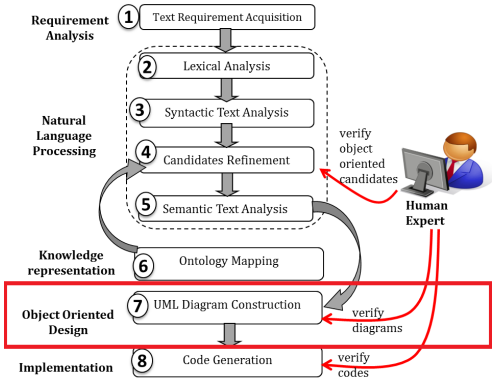

Fig. 1. Architecture of Use Case specification to Sequence Diagrams (UC2SD)

In this paper, we will only discuss the Object Oriented Design Module with the UML Diagram Construction task (see Fig. 1). A detailed discussion on UC2SD architecture can be obtained in our previous publication [5]. Participating actors/objects/classes, messages/ methods and attributes are mapped respectively with nouns, verbs and adjectives and are then translated into UML sequence diagram constructs. In the beginning, a system sequence diagram (SSD) will be produced, followed by a detailed sequence diagram. Fig. 2 shows the process flow and how both the diagrams are generated. The output from the NLP processor will be used to generate sequence diagrams. Each sequence diagram will be visualized not just for main success scenarios but also for alternative and exception scenarios.

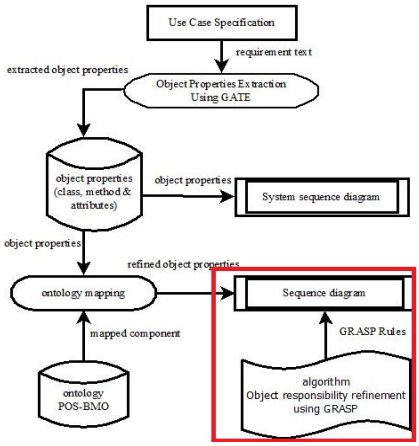

Fig. 2. Process flow of Use Case specification to Sequence Diagrams (UC2SD).

Following the generation of a System Sequence Diagram (SSD), detailed sequence diagrams can follow. As illustrated in Fig. 3, we assume the stereotypes: boundary, controller and entity classes. Boundary classes are those that interact with system actors; Entity Classes are those that represent objects in the system. Controller classes are those that mediate between Boundary and Entity classes, handling calls and passing the responsibility of responding to these to entity objects.

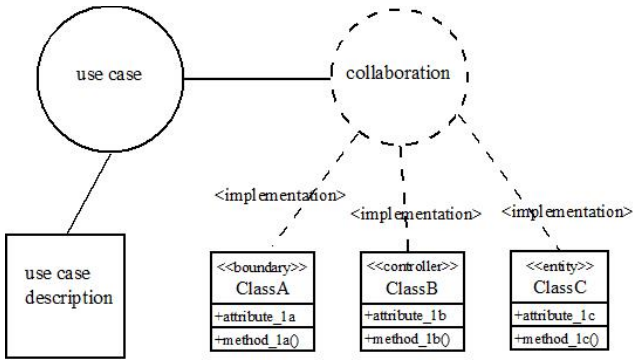

Fig. 3. Use case realisation in Sequence Diagram

To move to detailed sequence diagrams, we must consider how objects collaborate with other objects to implement system operation. To achieve this we have applied the General Responsibility Assignment Software Principles (GRASP) [19], including Creator, Information Expert, Controller, Low Coupling and High Cohesion principles.

There are used as follows:

- Creator determines which object should be responsible for creating another specific object;
- Information Expert determines which object should be responsible for a responsibility based on it having the necessary data to fulfil the responsibility;
- Controller determines which object should be the first to receive a message from an external actor
- Low Coupling is used to choose between objects for responsibility assignment, based on the degree of interaction between objects;
- High Cohesion is used to choose between objects for responsibility assignment, based on how internally related the assigned responsibility is in the case of each object.

The goal of applying these principles is to identify class responsibility which in turn establishes its collaboration.

IV. Demonstration And Analysis

A Use Case specification to Sequence Diagrams (UC2SD) generator was designed and constructed in order to demonstrate the approach. The use case specification template has been formulated, which includes the most important components in order to build the sequence diagram.

We used the processing resources that GATE [7][20] provides which are made available in the form of plug-ins. We also use 'A Nearly-New IE' (ANNIE) system [21] which supports a sentence splitter, tokeniser, morphological analyser, part of speech tagger, gazetter and orthomatcher. Aside from ANNIE, GATE makes it possible to use the Java Annotations Pattern Engine (JAPE) transducer [7] which provides a way to process text over specified annotations and to further identify patterns or entities in text. ANNIE relies on finite state algorithms and JAPE even supports an Ontology-API[22] which helps represent knowledge understanding in object relations.

Input text is from the use case provided by the user, which needs to be tokenized and split into sentences. In every information retrieval process, this is a common procedure. Each token (i.e. number, word, punctuation) is then assigned with Part-of-Speech (POS) tags where the grammars are based on Penn Treebank Tagset which applies the Hepple's Brill-style tagger [23]. Hence, a word that is found to be a 'stop word' will be eliminated (i.e.: a, maybe, the, etc…). This process is assisted by a morphological analyser which involves lemmatization or word stemming. Next, the JAPE transducer will trigger the grammar rule to identify and annotate objects and messages from the given Syntactic Rules (SR), attached in the previous publication [5]. The JAPE syntaxes have been developed for all of the SRs.

Thus, before the XML is produced, a frequency analysis step is carried out to produce frequency lists of overall word form. The selection of candidate classes are based upon the frequency of the nouns appearance, and the result will then be verified by the user. This so called object properties extraction are now used to construct a System Sequence Diagram (SSD).The SSD is a sequence diagram that shows the event interaction between external actors with the system object. Fig. 4 and 5 illustrates the Point of Sales (POS) system specification for process sale use case and SSD generated. This SSD describes:

- each method is labelled above the arrow;
- method parameters in brackets for each message;
- the message represented as solid arrows and returns represented as a dotted line arrow.

The SSD generation is a relatively straightforward task as it only involves the external actors, message flow and System object. However, more detailed system design can be derived with potential classes involving three common stereotypes (boundary, controller and entities). Thus, to construct a refined Sequence Diagram (rSD) as shown in Fig. 8, the classes selected from potential classes were finalised and responsibilities determined for objects within the system. To achieve this, we use the Point of Sales – Business Management Ontologies (POS-BMO) ontology [5][19] to map the extracted object properties extraction to appropriate objects.

First the tool adds all the entity objects from preceding ontology analysis and then it transforms some individual messages on the GRASP rules. Finally, it divides the System class into a Boundary class; a Controller class and Entity classes adjust the messages accordingly. All of this is currently done against a representation of the sequence diagrams in XML, via the Document Object Model (DOM) API.

In Creator, the pattern directs us to who should be responsible for creating a new instance of some class? According to Larman [6], using a Point of Sale system example, the Creator principle applied to 'Process Sale' use case is justified as follows:

- *SystemRegister* is responsible for creating the *Sale* object because the *SystemRegister* is used by the *Cashier* to ring in a new *Sale*.
- The *Sale* object is responsible for creating the *Payment* object, as *Payment* is only being made when a *Sale* is being made. Hence, the *SystemRegister* makes a *Sale* object which in turn makes a *Payment* object.

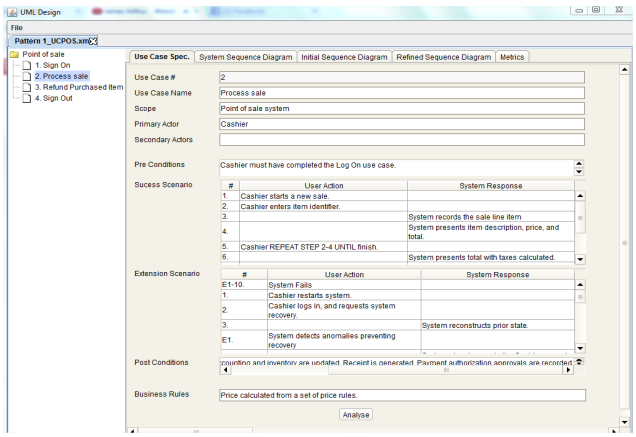

Fig. 4. UC2SD Automation of System Sequence Diagram (SSD) production

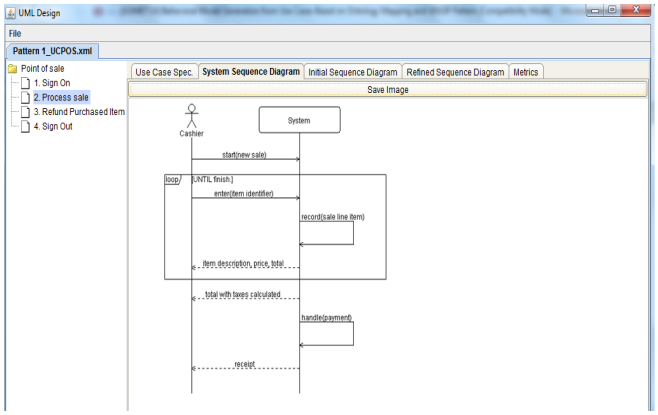

Fig. 5. System Sequence Diagram (SSD) of Process Sale use case generation

The Information Expert principle guides us to assign responsibilities to objects where the object becomes an expert for service if it has the ability or information to fulfil the obligations of that service. According to Larman [6], the Information Expert principle applied to the Process Sale use case is illustrated as follows:

i. *SystemRegister* is the Information Expert for the following services: *makeNewSale*, *enterItem*, *endSale*, and *makePayment*, as it has the requisite information on hand to fulfil these obligations.
ii. The *Sale* object is responsible for *getTotal*, *makePayment*, and *makeLineItem*. The picture should also include a call of *makeLineItem* to the *SalesLineItem* object.
iii. The *Product* object is responsible for providing its own *price*, so it has a function called *getPrice* for this activity.

The Controller pattern handles system operation messages between the actor and the first object in the domain layer. It is responsible for delegating tasks to other objects. In general, we are trying to put the methods in a class that has the most knowledge of how to implement the method. This class will serve as a Controller for other classes and in a way it minimizes the number of cross-dependencies between each class. We assumed that a Controller pattern can be identified with the controller stereotype class, detailed earlier. In one of the heuristic [23] we can identify a controller class for each use case. Again the use of the Controller principle applied to Process Sale use case is described as follows:

- *SystemRegister* is the controller class who is responsible for delegating messages flow from Cashier as the actor class.
- Before each message received by entity classes, *SystemRegister* will take control of the messages.

The GRASP principles of High Cohesion and Low Coupling are not hard-and-fast rules, but rather goals to be achieved in so far as possible, given other constraints, and also given their potential to sometimes conflict with each other. Finding an optimal design under GRASP thus involves a search through the solution space for the best solution. Such a search is not too complex if these are the only "soft constraints" we wish to optimize for. But if we wish to allow additional soft constraints to be specified in the future, then the complexity can grow very quickly. Fortunately, there are some off-the-shelf open source tools handling such complex directed searches. The so called tool used is the Drools Planner [25][26]. With any such planner, the soft constraints need to be quantified so that competing solutions can be compared. This quantification is called a metric or a fitness function. Quantifying Coupling is straight-forward. Quantifying Cohesion is not so easy, because it requires knowledge of the semantics of functions (whether two functions are "related" or "unrelated"). Our system will have to rely on an ontology to provide this semantic information.

Fig. 6 and Fig. 7 show the metric evaluation for Coupling and Cohesion in both initial and refined sequence diagrams. The counts in the table on the metrics tab are combined into a scalar metrics value for Coupling, and likewise for Cohesion. Metrics for both the initial Sequence Diagram (iSD) and refined Sequence Diagram (rSD) are shown to show how the refinement is chosen and how it will decrease Coupling and increase Cohesion. Below are the formula used to calculate the metric Coupling on both iSD and rSD:

$$\text{average of message flows} = \frac{\sum \text{number of message flows}}{\text{number of classes}}$$

$$\text{average of class interactions} = \frac{\sum \text{number of message class interaction}}{\text{number of classes}}$$

The higher the average scores, the higher possibility the diagrams will be chosen. In some cases, iSD will produce higher scores than rSD which means the quality of the sequence diagram is better. To illustrate how the metrics measurement for iSD and rSD is being applied, referring to figures 6 and 7:

i. If the design shown currently in the rSD tab produces worse or lower metrics count than the one in iSD, the tool will not accept the design currently as the rSD. It would either leave the design unchanged from iSD, or possibly produce some different design.
ii. The idea of refinement is mainly to reduce the coupling but if it also increases Cohesion, then we can justify the increase in Coupling as a trade-off.

The result of the metric Cohesion shows none of the classes has a different Cohesion score for the rSD than for the iSD. So the only metric that's different at this stage is the Coupling, where iSD is better than the rSD. The quality of the sequence diagram is evaluated in terms of a set of measures: Completeness, Correctness, and BCE (the Boundary/ Controller/ Entity principle) Consistency[27]. To evaluate the quality of sequence diagrams produced by participants, we will make use of sequence diagrams given in textbooks or other resources. In the case of none of the resources being available, help from Object Oriented experts are needed.

The completeness of a sequence diagram ($SD^{complete}$) is calculated as the average of the completeness of the messages, interaction uses, and combined fragments contained in the sequence diagram.

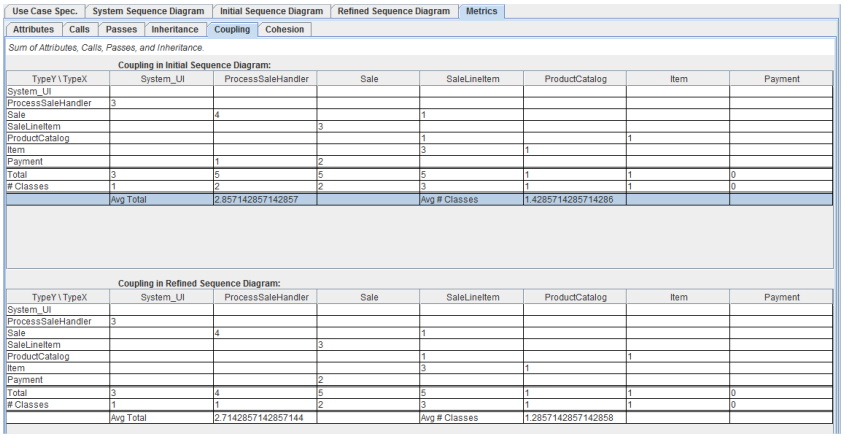

Fig. 6. Metric coupling in iSD and rSD

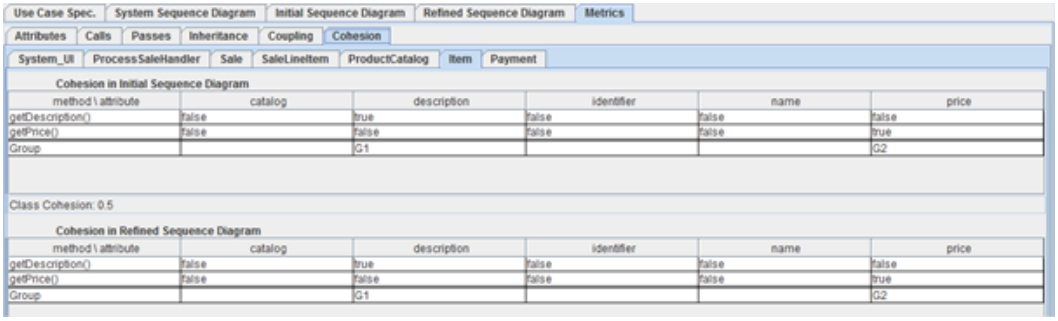

Fig. 7. Metric Cohesion in iSD and rSD – Item Class

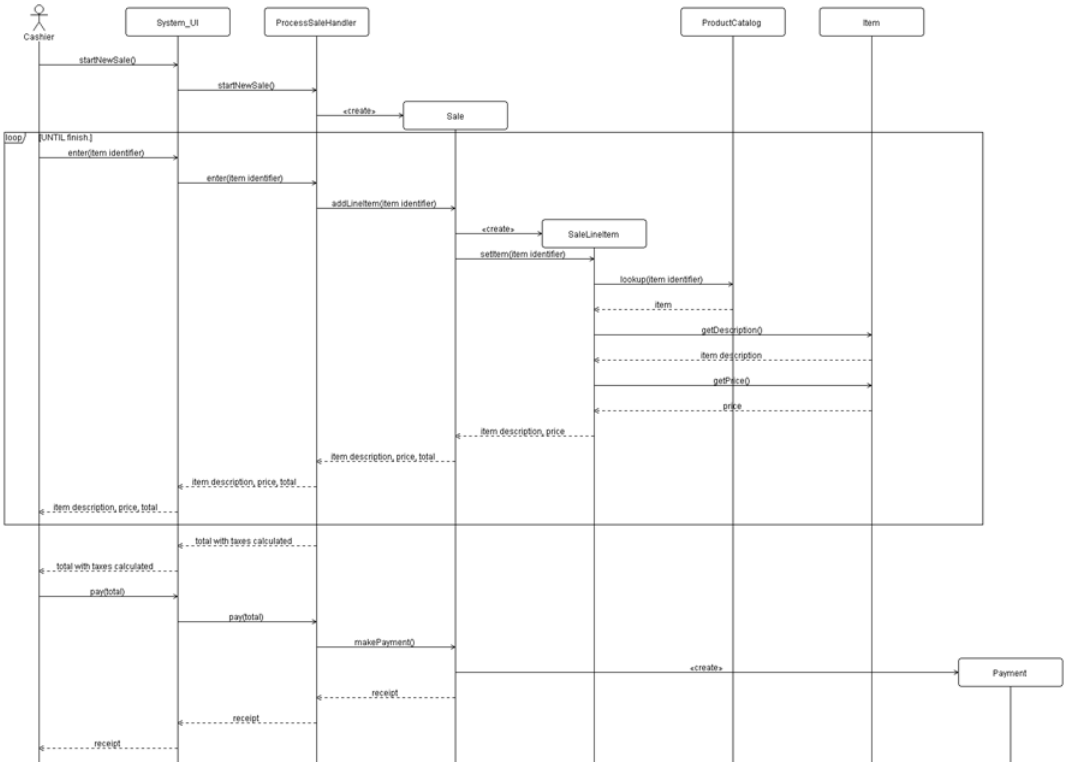

Fig. 8. Refined Sequence Diagram

The correctness of the sequence diagram ($SD^{correct}$) is evaluated as the average of the correctness of the messages, interaction uses, and combined fragments of the sequence diagram.

The rationale is that the message is one of the most important elements of sequence diagrams and SD Completeness, and SD Correctness indicate the overall completeness and correctness of a sequence diagram, including the completeness and correctness of its interaction uses and combined fragments. Therefore, it is not necessary to report on the completeness and correctness of interaction uses and combined fragments separately. Meanwhile, the BCE measurement is evaluated through a number of messages passed through the class stereotypes. This interaction will not allow the Boundary classes to interact with the Entity classes. The Controller becomes the mediator to manage the interaction between the classes.

V. Discussion And Conclusion

Each of the GRASP patterns that have been applied to identify the class relationship and collaboration has their own generic task and uses. Creator, Controller, and the Information Expert can be evaluated by using the BCE measurement and rules applied. The SSD is been generated before the iSD and rSD can be visualised where the classes are categorised into common stereotypes of BCE structures. Then, through coupling and cohesion class metric measurement, the interaction between classes was measured to improve the quality of the sequence diagram. By applying GRASP on CRA it was concluded that the solution need not always be the best but it will be optimal. Possibilities of deriving the best solution are still under study. In the object-oriented software system, every class should have set of responsibility, and it should be allocated optimally. We can find the optimal fitness function or metric calculation for each software component.

In automatically producing behavioural models from text, we have to identify the proper object, classes, attributes relationships and so forth for building a System Sequence Diagram and Refined Sequence Diagram (rSD) by applying rules based on GRASP Principles. We have successfully developed a computerized support for CRA to provide a cognitive toolset to help designers and developers on the analysis and design of object-oriented software.

Acknowledgment

The authors gratefully acknowledge Ministry of Higher Education (MOHE) Malaysia, as part of the first author's PhD studies scholarship. Furthermore, the authors also recognise the funding support for the conference provided by the School of Electronics, Electrical Engineering and Computer Science of Queen's University Belfast.

References

[1] M. E. Fayad, H. Hamza, and H. Sanchez, "A pattern for an effective class responsibility collaborator (CRC) cards," *Proc. Fifth IEEE Work. Mob. Comput. Syst. Appl.*, pp. 584–587.

[2] D. Svetinovic, D. M. Berry, and M. Godfrey, "Concept identification in object-oriented domain analysis: why some students just don't get it," *13th IEEE Int. Conf. Requir. Eng.*, pp. 189–198, 2005.

[3] M. El-Attar and J. Miller, "Constructing high quality use case models: a systematic review of current practices," *Requir. Eng*, vol. 17, pp. 187–201, 2012.

[4] R. M. Noorullah, "Grasp and GOF Patterns in Solving Design Problems," *Int. J. Eng.*, vol. 1, no. 3, pp. 196–205, 2011.

[5] N. Jali, D. Greer, and P. Hanna, "Behavioral Model Generation from Use Cases Based on Ontology Mapping and GRASP Patterns," in *26th International Conference on Software Engineering and Knowledge Engineering*, 2014.

[6] H. Cunningham and D. Scott, "Software Architecture for Language Engineering," *Nat. Lang. Eng.*, vol. 10, no. 3–4, pp. 205–209, Sep. 2004.

[7] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan, "GATE: A framework and graphical development environment for robust NLP tools and applications," in *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002, pp. 168–175.

[8] M. Bowman, L. C. Briand, and Y. Labiche, "Multi-Objective Genetic Algorithms to Support Class Responsibility Assignment," in *Software Maintenance 2007 ICSM 2007 IEEE International Conference on*, 2007, pp. 124–133.

[9] C. Larman, *Applying UML and Patterns - An Introduction to OO Analysis and Design and the Unified Process*. Addison-Wesley Professional, 2004, p. 736.

[10] R. Wirfs-Brock and B. Wilkerson, "Object-oriented design: a responsibility-driven approach," *ACM SIGPLAN Notices*, vol. 24. pp. 71–75, 1989.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. 1995, p. 395.

[12] M. Bowman, L. C. Briand, and Y. Labiche, "Solving the Class Responsibility Assignment Problem in Object-oriented Analysis with Multi-Objective Genetic Algorithms," *Work*, pp. 1–48, 2010.

[13] G. Glavas and K. Fertalj, "Metaheuristic Approach to Class Responsibility Assignment Problem," *Electr. Eng.*, pp. 591–596, 2011.

[14] G. Glavas and K. Fertalj, "Solving the Class Responsibility Assignment Problem Using Metaheuristic Approach," *J. Comput. Inf. Technol.*, vol. 19, no. 4, pp. 275–283, 2011.

[15] S. Acharya, "The Process of Information Extraction through Natural Language Processing," *Int. J. Log. Comput.*, vol. 1, no. 1, pp. 40–51, 2010.

[16] R. Collobert, J. Weston, and M. Karlen, "Natural Language Processing (almost) from Scratch," vol. 1, pp. 1–34, 2000.

[17] P. R. Kothari, "Processing Natural Language Requirement to Extract Basic Elements of a Class," *Int. J. Appl. Inf. Syst.*, vol. 3, no. 7, pp. 39–42, 2012.

[18] J. Nicolás and A. Toval, "On the generation of requirements specifications from software engineering models: A systematic literature review," *Inf. Softw. Technol.*, vol. 51, no. 9, pp. 1291–1307, 2009.

[19] C. Larman, *Applying UML and Patterns*, 2nd ed. Prentice Hall, 2001.

[20] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham, "Evolving GATE to meet new challenges in language engineering," *Nat. Lang. Eng.*, vol. 10, no. 3–4, pp. 349–373, 2004.

[21] M. Dimitrov, "A Light-weight Approach to Coreference Resolution for Named Entities in Text," University of Sofia "St. Kliment Ohridski", Bulgaria, 2002.

[22] N. F. Noy and D. L. Mcguinness, "Ontology Development 101: A Guide to Creating Your First Ontology," *Development*, pp. 1–25, 2000.

[23] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering (Using UML, Pattern and Java)*, 3rd ed. New York, US: Pearson Higher Education, 2010.

[24] Jenz and P. Gmbh, "Business Management Ontology (BMO) Version 1.0," Germany, 2004.

[25] D. V. . Weppenaar and H. J. Vermaak, "Solving planning problems with Drools Planner," *Koers Bull. Christ. Scholarsh.*, vol. 10, no. 1, pp. 91–109, 2011.

[26] T. J. Drools, "Drools Planner User Guide."

[27] N. Kama, T. French, and M. Reynolds, "Design Patterns Consideration in Class Interactions Prediction Development," Sci. Technol., vol. 28, no. 1, pp. 45–64, 2011.